

Bachelor-Thesis

in

Medieninformatik

Konzeption, Implementation und Evaluation von Netzwerkkomponenten für eine auf die Lehre spezialisierte Game-Engine

Referent: Prof. Jirka Dell‘Oro-Friedl

Korreferent: Prof. Dr. Ruxandra Lasowski

Vorgelegt am: 31.8. 2100

Vorgelegt von: Falco Böhnke

250100

Im Großacker, 28

79252, Stegen

falco.boehnke@hs-furtwangen.de

Abstract 100-120wort

Im Rahmen dieser Bachelorarbeit wurden Komponenten für die Netzwerkkommunikation innerhalb des auf Electron basierenden Editors „Fudge“ konzipiert und entwickelt. Dafür werden zuerst die Anforderungen, vorgegeben durch Electron und Fudge selbst, festgelegt.

Im Anschluss daran wird erläutert, welche Varianten von Netzwerkkommunikation in heutigen Videospielen üblich sind, welche Protokolle für die Datenübertragung verfügbar sind und welche existierenden Webtechnologien verfügbar sind um mithilfe der jeweiligen Protokolle zu kommunizieren.

Nach Auswahl der zu verwendenden Technologien wird die Konzeption erläutert und die Komponenten umgesetzt und ihre Funktionsweise mithilfe von UML-Diagrammen und Aktivitäts-Diagrammen dargelegt.

Zum Abschluss wird auf die Zukunftsaussichten und etwaige Möglichkeiten der Weiterentwicklung eingegangen und eine Schlussfolgerung über die Entwicklung von Netzwerkkomponenten für einen didaktischen Spiele gezogen.

Inhaltsverzeichnis

[Abstract 3](#_Toc17290126)

[Inhaltsverzeichnis 5](#_Toc17290127)

[Abbildungsverzeichnis 7](#_Toc17290128)

[Abkürzungsverzeichnis 9](#_Toc17290129)

[Einleitung 1](#_Toc17290130)

[Fudge – Didaktischer Spieleeditor 3](#_Toc17290131)

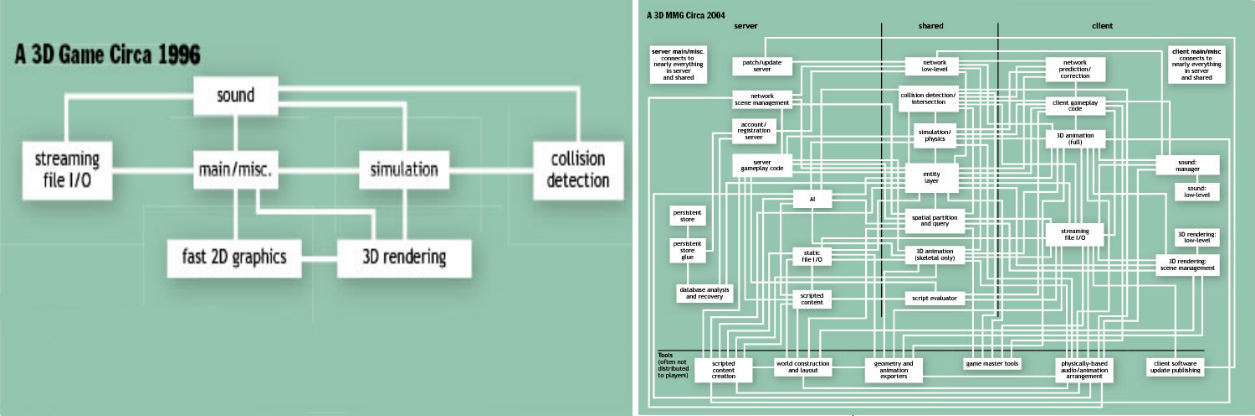
Abbildungsverzeichnis

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Abkürzungsverzeichnis

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

1 Einleitung

Die Videospielindustrie ist in den letzten Jahren massiv gewachsen. Allein der Markt für Videospiele in den Niederlanden wuchs von 518 Millionen Euro im Jahr 2012 auf 1.65 Milliarden Euro im Jahr 2019 an, mit vergleichbaren Tendenzen, mit Wachstumszahlen von 11% über die letzten 10 Jahre, weltweit (Wijman 2018) (Kazimier 2017). Der starke Anstieg der Umsatzzahlen hat zu einem gesteigerten Konkurrenzkampf innerhalb des Videospielmarktes geführt. Innovationen sind unerlässlich geworden und so sehen sich Entwickler mit immer komplexeren Projektanforderungen konfrontiert. Insbesondere im Bereich der Netzwerkkommunikation sind die Anforderung an hoch performante Lösungen gestiegen.

**Abb. 1:** Vergleich der möglichen Komponenten für Videospiele aus den Jahren 1996 und 2004 in Anlehnung an Blow (2004)

Gameengines, Entwicklungsumgebungen, die für die Produktion von Videospielen optimiert sind, finden daher verstärkt Einsatz um Entwickler die Arbeit zu erleichtern und komplexe Komponenten verwendungsfertig bereit zu stellen. Diese fertigen Komponenten sollen durch Automation und Abstraktion Entwicklern fundamentale Arbeiten abnehmen und so Kapazitäten für Optimierung befreien. Doch die Undurchsichtigkeit der abstrahierten Systeme und die oftmals komplizierte Struktur der Game-Engines macht es Entwicklern in Ausbildung schwer die grundlegende Funktionsweise der einzelnen Komponenten eines Videospiels zu erfassen und zu erlernen. So sind Entwickler gezwungen statt grundlegender Konzepte die Funktionsweise ihrer ausgewählten Entwicklungsumgebung zu lernen. Ist dann ein Wechsel auf eine andere Entwicklungsumgebung notwendig, müssen Entwickler sich erneut an die Umgebungsentwicklung anpassen, wobei dort ähnliche Komponenten mitunter andere Funktionsweisen haben. Besonders der Mangel einer von Grund auf für die Lehre konzipierten Game-Engine verschärft diese Problematik weiter.

Um diese Lücke zu füllen wurde das Projekt ‚Fudge‘ von Prof. Jirka Dell‘Oro-Friedl ins Leben gerufen. Fudge ist eine Game-Engine und ein Editor, der die Strukturen und Prozesse eines Videospiels offen legt und Entwickler dennoch mit grundlegenden Funktionalitäten versorgt.

In dieser Arbeit wurden Netzwerkkomponenten entwickelt, die Entwicklern in Fudge die Möglichkeit geben vernetzte Spiele zu entwickeln, ohne die notwendigen Komponenten von Grund auf selbst zu schreiben. Dazu wird zuerst der Entwicklungsrahmen mit Fudge und Electron dargelegt. Anschließend wurden die in modernen Spielen üblichsten Methoden zur Kommunikation zwischen Spielinstanzen ermittelt und kurz erläutert. Dazu gehört ebenfalls ein kurzer Exkurs in modernen Netzwerkprotokollen die in der Netzwerkkommunikation üblich sind und daher relevant für die Umsetzung der Netzwerkkommunikation in Fudge.

Aus diesen Informationen erschließen sich die Anforderungen an die Netzwerkkomponenten und die Aufgaben die sie erfüllen können müssen, um für die Entwicklung von Spielen in Fudge sinnvoll verwendet werden zu können.

Unter Berücksichtigung der Rahmenbedingungen und Anforderungen werden dann die möglichen Technologien ermittelt, die bei der Entwicklung verwendet werden können. Diese werden evaluiert und die passenden werden im weiteren für die Entwicklung verwendet

Darauf folgend wird die Umsetzung der einzelnen notwendigen Netzwerkkomponenten dargelegt und ihr Aufbaue mithilfe von Unified Modeling Language Diagrammen dargestellt, sowie ihre Funktionsweise und Ablauf beispielhaft mithilfe von Aktivitätsdiagrammen dargestellt.

Abschließend werden die Ergebnisse der Evaluation und Entwicklung zusammengefasst und ein Fingerzeig für eine mögliche Weiterentwicklung der Komponenten gegeben.

2 Rahmenbedingungen – Theoretische Grundlagen

In diesem Kapitel werden die Rahmenbedingungen für die Entwicklung der Netzwerkkomponenten dargelegt, welche Einschränkungen und Besonderheiten existieren und welche besonderen Anforderungen durch Fudge und Electron entstehen.

Fudge

Fudge ist eine open Source die Game-Engine und Spieleditor für die Entwicklung von zwei dimensionalen und drei dimensionalen Sipelen kombiniert und besonders augenmerk auf didaktisch sinnvolle Strukturen sowie menschenlesbaren Quellcode legt. Der Name gründet sich dabei aus dem vollen Titel der Software: „Furtwangen University Didactic Game Editor“. Als Game-Engine übernimmt Fudge dabei grundlegende Aufgaben wie die visuelle Repräsentation des Spielverlaufs sowie dem allgemeinen Ablauf, Audiowiedergabe und Animationen. Dazu kommt, auf seiten des Spieleditors, eine einfache Darstellung der Komponenten die in einem gegebenen Spielzustand verwendet werden können, sowie deren Werte. Fudge ähnelt in dieser Hinsicht stark den gebräuchlicheren kommerziellen Game-Engines wie Unity, die Unreal Engine und Frostbtte, die ebenfalls Editor und Engine kombinieren. Gegenüber diesen kommerziellen Produkten ist Fudge allerdings auf die Anwendung in der akademischen Ausbildung spezialisiert und hat dadurch in diesem Bereich klare vorteile. Datenformate sind allgemein gültig und menschenlesbar gestaltet, das bedeutet der Quellcode soll sich durch geschickte Namensgebung und klare Strukturen selbst erklären können. Außerdem wird so die Versionskontrolle vereinfacht, da eine Integration mit Anbietern wie Github oder Gitlab von grundauf eingeplant und möglich ist. Ein weiterer wichtiger Punkt ist, dass Fudge sich auf das prinzip von „Composition over inheritance“(Coi) stützt. Dem CoI-Prinzip folgend werden Komponenten kreeirt die wiederverwendbar sind und aus denen sich andere Klassen zusammensetzen lassen. Außerdem reduziert das CoI-Prinzip die Notwendigkeit Gemeinsamkeiten zwischen Klassen zu finden, um sie in sinnvolle Familienbäume zusammenzufassen.

Fudge basiert in seiner grundlegenden Struktur auf Webtechnologien, namentlich HTML, CSS, JavaScript und TypeScript, NodeJS und Electron in Kombination mit der Browserumgebung Chromium. Diese erlauben in Kombination mit Elektron für einen auf den geläufigsten Desktopgeräten verwendbaren Editor, dessen gepackte Endprodukte crossplattformkompatibel sind, das bedeutet auf verschiedenen Betriebssystemen, Geräten und sogar Browsern nutzbar sind.

Besonderes Augenmerk liegt im Rahmen dieser Bachelorarbeit auf TypeScript, NodeJS und Electron, da Entscheidungen über Kompatibilität mit Einschränkungen und Möglichkeiten getroffen werden müssen, die von diesen Technologien ausgehen. Im Folgenden wird kurz erläutert, was die einzelnen Technologien sind und an Möglichkeiten bieten, und inwiefern sie die Entwicklung der Netzwerkkomponenten für Fudge beeinflussen.

JavaScript und TypeScript

JavaScript ist, wie der Name impliziert, eine Skriptsprache und eine der Kerntechnologien auf denen das World Wide Web aufbaut. Eine Skriptsprache ist dabei explizit designed um auf ein existierendes System oder eine vorhanden Entität aufzubauen. Als solches folgt JavaScript dem international anerkannten ECMAScript Standard. Dieser Standard besagt welche Typen, Werte, Objekte, Eigenschaften, Funktionen und Programm Syntax sowie Semantic von JavaScript angeboten und unterstützt werden müssen

JavaScript ist zudem eine interpretierte, keine kompilierte Programmiersprache Kompilation bedeutet, dass Quellcode von einer Software, dem Compiler, in maschinencode übersetzt wird. Der Compiler liefert als Resultat ein Programm in Maschinencode zurück. Ein Interpreter dagegen liefert das Ergebnis eines Quellcodes zurück, also das Ergebnis einer Berechnung. Allerdings muss beachtet werden dass die Grenzen zwischen interpretierten und kompilierten Sprachen immer mehr verwischen, da mit der V8 JavaScript Engine auch Javascript zu Maschinencode kompiliert werden kann. Dies bringt Geschwindigkeitsvorteile mit sich.

Da JavaScript in jedem modernen Browser nutzbar ist können angehende Entwickler schnell erste Erfolge erzielen. Zudem lassen sich komplexe JavaScript Abläufe Stück für Stück programmieren und im Browser auf ihre funktionalität testen. Dies erleichtert den Einstieg erheblich und senkt so die Lernkurve stark ab.

Außerdem existieren viele Fallstricke anderer Programmiersprachen in JS nicht, unter anderem Einschränkungen durch Typisierung und komplexe Sturkturen durch multiple Vererbung, die Erfahrung und solide Planung voraussetzen.

Ein weiterer Vorteil, der JavaScript als Grundbaustein des World Wide Web zementierte, ist die Codeausführung auf Seiten des Clienten. Dabei liefert ein Server lediglich den Quellcode an dne Clienten, das Programm wird jedoch im Browser des Clienten ausgeführt.So lassen sich Daten bis zu einem gewissen Grad validieren, ohne dafür Serverressourcen verwenden zu müssen. Dies geschieht auf Kosten der Sicherheit. Browser injezierte JavaScript Programme können Daten auslesen und verschicken und sogar Schäden an den ausführenden Geräten selbst verursachen. Außerdem erschwert die fehlende Typisierung und der nicht existente Debugger, Software die Fehler im quellcode erkennt und ausliest, die Wartung ungemein und führt zu Laufzeitfehlern, die nur durch zeitintensive Fehlersuche behoben werden können. Dazu kommt, dass die Verwendung von Objekt-orientierten Prinzipien durch die fehlende Typisierung und ausschließlich einfache Vererbung viel Kreativität erfordert.

Dennoch erfreut sich JS sehr großer Beliebtheit und ist heute, 2019, der defacto Standard für moderne Browser, unterstützt von Google, Facebook und Mozilla.

Einige der Nachteile von JavaScript lassen sich jedoch mithilfe von Typescript beheben und umgehen. TypeScript ist eine von Microsoft entwickeltes, typisiertes Superset von JavaScript. Quellcode wird in TypeScript geschrieben und der Typescript eigene Compiler wandelt den Code in funktionierendes JavaScript um. Durch die nun vorhandene Typisierung und einen TypeScript eigenen Debugger sind Fehler jedoch sbereits im Quellcode leicht ersichtlich. Klassen, Objekte und ihre Funktionen lassen sich deklasieren und festlegen; sind eindeutig definiert und können während der Laufzeit nicht ungewollt verändert oder überschrieben werden. So lassen sich objektoriente Programmierprinzipien erfolgreich und problemlos anwenden, der Quellcode ist dadurch übersichtlich strukturiert, die Wartbarkeit ist vereinfacht und der Quellcode lässt sich mit allgemeingültigen Modellen, wie der Unified Modelling Language, darstellen und dokumentieren. Die Beliebtheit von TypeScript steigt ob dieser Vorteile stetig an und ersetzt mehr und mehr Entwicklung mit reinem JavaScript.

Für die Entwicklung in Fudge wird ausschließlich TypeScript verwendet, da der resultierende, noch nicht kompilierte Quellcode sich menschenlesbar gestalten lässt, ein Debugger zur Verfügung steht und sich der Code durch die Möglichkeiten der Objektorientierten Programmierung in modulare Komponenten aufsspalten lässt.

Node.js

Node.js ist eine Laufzeitumgebung die alle Komponenten beinhaltet die für die Ausführung von JavaScript Code notwendig sind. Node.js basiert, wie JavaScript selbst, auf der V8 JavaScript Runtime Engine und wird zu maschinencode übersetzt. Node.js ist durch seine Autarkität geeignet, um in Serverumgebungen zu arbeiten. Im Gegensatz zu Javascript, dessen Event-Schleife nur auf einem Thread arbeitet, arbeitet Node.js jedoch asynchron und kann mehrere Threads gleichzeitig verwalten, ohne dass andere Funktionen blockiert werden. Eine entscheidende Eigenschaft die bedeutet das nicht jede Anfrage nacheinander, sondern gleichzeitig bearbeitet werden kann. Dies ist besonders kritisch in einer Serverumgebung da ansonsten jede Anfrage nacheinander einzeln bearbeitet werden muss, und sich so Anfragen gegenseitig blockieren. Durch die asynchrone Arbeitsweise kann Node.js nicht-blockierende Input/Output Requests anbieten.

Node.js erfreut sich dabei ebenso wie JavaScript großer Beliebtheit wodurch viele Bibliotheken und Module zur freien Verfügung stehen. Über den node.js eigenen Package-Verwalter „npm“ können diese problemlos abgerufen, installiert und integriert werden. Dies macht Node.js kompatibel mit den meisten geläufigen Programmiersprachen, APIs und Programmen.

Node.js bietet außerdem grundlegende Komponenten der Netzwerkkommunikation, auf die im späteren eingegangen wrid.

Electron

Electron basiert auf dem Projekt Atom Shell das im Jahr 2013 begann und im Jahr 2015 zu Electron umbenannt wurde. Ursprünglich war es als cross-platform text editor gedacht, in welchem der Nutzer mit Webtechnologien wie Javascript CSS und HTML arbeiten kann.

Electron kombiniert Chromiums rendering library, die ein open-source Fundament für Google Chrome darstellt, mit Node.js und der V8 JavaScript Engine in einer einzigen Laufzeitumgebung. Dadurch können Programme die in JavaScript oder Node.js geschrieben werden ohne Einschränkungen verwendet werden und neue Programme auf Basis von Node.js, HTML/CSS und JavaScript erstellt werden. Diese Programme sind cross-plattform kompatibel und können sowohl für Desktop, Mobile und Browserumgebungen ausgegeben werden. So können Webentwickler auch vollwertige Desktopsoftware und sogar Applikationen für Smartphones erstellen, ohne von ihren bekannten Technologien abweichen zu müssen. Wichtig ist jedoch zu beachten, dass Electron auf einer Browserumgebung aufbaut und dementsprechend Einschränkungen im Bereich des Dateizugriffs aufweist, sowie die browserüblichen Sicherheitssysteme für die Netzwerkkommunikation.

Netzwerkprotokolle

Udp

Tcp

ftp

Heutige Spiele

Clientserver

Eine der ältesten und meist genutzten Varianten der Netzwerkkommunikation in Spielen stellt die Client-Server Struktur dar. Dabei gibt es einen dedizierten Server, der als Dreh und Angelpunkt für alle Clientinteraktionen mit dem Spiel und miteinander dient. Der Server arbeitet Anfragen ab, synchronisiert Spielstände, validiert Eingaben der User und leitet relevante Informationen an die Clients weiter. Dies hat sich insbesondere bei rasanten Spielen wie Multiplayer-Online-Shootern wie beispielsweise Apex Legends, Overwatch und Fortnite durchgesetzt, da die Server in einer stabilen Infrastruktur aufgebaut werden und ganztägig betrieben werden können und so stetige und vor allem hochperformante Spielerlebnisse bieten können. Latenz und Verbindungsabbrüche sind hierbei üblicherweise vom Client ausgelöst und müssen so seitens der Anbieter nicht gewartet oder anderweitig unterstützt werden. Zudem erlaubt eine Client-Struktur die Kontrolle des Spielerlebnisses und den Ausschluss von Spielern, da die Spiele ausschließlich dann funktionieren, wenn sie Kontakt zu einem der Spieleserver haben. Gleichzeitig erfordert diese Art der Netzwerkkommunikation eine solide Serverinfrastruktur, da ansonsten das Spielerlebnis für alle Spieler gleichermaßen unter Lags, Verzögerungen die durch langsame Übertragung von Daten entstehen, leidet.

Peer To Peer – Meshverbindungen

Seltener genutzt wird das Peer-To-Peer Verbindungsverfahren. Hierbei gibt es keinen dedizierten Server. Stattdessen verbinden sich alle Clients miteinander und erzeugen ein sogenanntes Mesh. Spielstände werden zwischen den Clienten synchronisiert und Eingaben gehen von den initierenden Clienten in das Netzwerk, werden von den empfangenden Clienten verarbeitet und angezeigt. Dies erlaubt es Serverfreie Multiplayerspiele zu erschaffen und so die Kosten für die Bereitstellung und Wartung der Serverinfrastruktur zu umgehen. Besonders bei kleineren Studios und einzelnen Entwicklern kann dies ein entscheidendes Kriterum sein. Peer To Peer Verbindungen sind dafür fehleranfällig, da die Verbindungsqualität aller Clients oftmals von jedem andere Client abhängig ist. So kann eine schlechte Verbindung das Spielerlebnis vieler einschränken, auch wenn dies durch geschickte Planung und entsprechende Kontrollsysteme reduziert werden kann. Außerdem gibt es keine zentrale Validierung der ins Netzwerk gesendeten Daten. Dadurch sind Peer to Peer Meshes anfällig für Dateninjektionen oder andere Betrügereien, die einem bestimmten Spieler massive Vorteile bieten. Zu guter letzt lassen sich so keine größeren Multiplayerspiele realsieren, da die Organisation eines Peer to Peer Meshes eine Vielzahl von Einzelverbindungen erzeugt. In einem 20-Spieler Mesh muss jeder Client 19 Verbindungen verwalten, wodurch die Hardwareanforderungen stark ansteigen.

Unechtes „Peer to Peer”

Bei unechtem Peer to Peer handelt es sich eigentlich um eine Client-Server Variante. Hierbei werden Clients üblicherweise per Matchmaking oder Lobbies zusammengeführt. Der Matchmaking/Lobbyserver organisiert dann die Verbindungen, indem er einen Client auswählt der als „Host“, also Client und Server gleichzeitig, fungiert und die Rolle des Servers übernimmt. Anschließend verbinden sich alle Clienten innerhalb der Lobby mit diesem Server, einschließlich dem Hostspieler selbst. In der Funktionsweise agiert diese Verbindungsart dann gleich wie eine normale Client-Server Verbindung. Dies erlaubt es dem Anbieter nur einen Server für das Matchmaking zu betreiben. Die Belastung dessen ist wesentlich geringer als bei einer reinen Client-Server Struktur und Latenzen zu diesem Server spielen keine Rolle. So können selbst schwache Server-Infrastrukturen den Anforderungen gerecht werden. Auf der Seite des Spiels hat der Host, der sich selbst als Spieler mit dem lokalen Server verbindet, stehts eine Verzögerung von 0ms. Dies bietet einen gewaltigen Vorteil gegenüber den anderen Clients. Außerdem ist diese Verbindungsart, ähnlich wie reines Peer to Peer, anfällig für Hosts mit schlechter Verbindung. Dazu muss ein eventuelles Beenden des Spiels oder ein Verbindungsabbruch seitens des gewählten Hosts berücksichtigt und entsprechende Ausnahmebehandlung programmiert werden. Üblicherweise bekannt ist diese als „Host Migration“: Dabei wird ein neuer Host bestimmt, der die Aufgaben des alten übernimmt. Anschließend werden die Spielstände erneut synchronisiert und das Spiel wird fortgesetzt. Dies führt zu teils sekundenlangen Unterbrechungen.

3 Methodik

Im letzten kapitel wurden die Voraussetzungen dargelegt die Technologien erfüllen müssen um für die Entwicklung von Netzwerkomponenten in Frage zu kommen. Im folgenden werden die möglichen Kandidaten vorgestellt und auf ihre Eignung geprüft.

Auswahl der kompatiblen Technologien

Darstellung als grafik zeitstrahl wo inkompatible sachen dann runterfallen sozusagen oder eher plus minus rot grün tabelle

Tech 1

Tech 2

Tech 3

Entwicklung

Komponenten

Komp 1

Komp 2

Komp 3

Komp 4

4. Praktische Anwendung

Neue Nachrichten abfangen

Clientenlogik einbauen

Authoritative Commands strukturieren

Ids für netzwerkobjete vergeben

5. Diskussion und Ausblick

Im Rahmen dieser Bachelorarbeit wurden funktioniernde Netzwerkkomponenten für die Game-Engine Fudge konzipiert und implementiert. Das Ergebnis zeigt, dass entsprechend der Spezifikationen von WebRTC und Chromium, UDP Kommunikation direkt zwischen Peers möglich ist, und sogar eine Serverstruktur aufgebaut und in Electron implementiert werden kann. Die Verwendung von Typescript hat die Fehlersuche stark vereinfacht und es erlaubt, einfache Strukturen zu erstellen, die Skalierbarkeit, Wartbarkeit und Modularität gewährleisten Dies zeigt auch eine gewisse Flexibilität in der Herangehensweise. Die hier erstellten Komponenten könnten anderweitig umgesetzt werden, auch wenn die Kommunikation mit dem UDP-Protokoll in einer Browserumgebung ausschließlich über webRTC erfolgen kann.

In dieser Arbeit wurde dabei die grundlegende Erstellung von Kommunikationswegen über UDP zwischen zwei Clienten, seien beide Peers oder sei es eine Client-Server Struktur, fokussiert. Die Kombination von Webtechnologien mit Servertechnologien im Rahmen einer Browserumgebung hat aufwendige Fehlersuche notwendig gemacht. Die Kommunikation über UDP innerhalb einer modernen Browserumgebung ist aus Sicherheitsgründen eigentlich unmöglich gemacht. Dadurch sind entgegen der Erwartungen Algorithmen zur Serverseitigen-Prediction, dem Austausch von Gamestates und weitläufigen Lobbysystemen im Rahmen der Arbeit nicht möglich gewesen.

Gleichzeitig bieten sich die Komponenten für weiterführende Arbeiten an, beispielsweise um serverseitige Prediction zu konzipieren und umzusetzen, Lobbysysteme zu entwickeln und serverseitige Gamestate Validierung zu implementieren. Zudem können potentiell spezialisierte Netzwerkprotokolle entwickelt werden, die genau auf die didaktischen Anforderungen von Fudge zugeschnitten sind und so die Abhängigkeit von UDP umgehen. Außerdem können optimierte Kommunikationswege erarbeitet werden, die die Latenz bei der Kommunikation zwischen Clients und Server weiter reduziert.

Die für diese Arbeit verwendeten Technologien sind weltweit anerkannt und viel genutzt, beispielsweise JavaScript und Chromium, beide unterstützt durch den Technologiekonzern Google, TypeScript entwickelt von Microsoft, und Electron dass auf Technologie von Microsoft basiert. So besteht für Fudge keine Gefahr in näherer Zukunft obsolete Webtechnologien oder Komponenten deren Kompatibilität nicht mehr gegeben ist zu beherbergen.

5.2 Weiterentwicklung der Kernkomponenten

Wie können die komponenten selbst weiterentwickelt werden z.b, durch

**server prediction,**

**lazy loading,**

**nachrichten validierung,**

**werte validierung**